ARMY RESEARCH LABORATORY

Android REST Client Application to View, Collect, and Exploit Video and Image Data

by Somiya Metu and Robert P. Winkler

ARL-TR-6639 **September 2013**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-6639 **September 2013**

Android REST Client Application to View, Collect, and Exploit Video and Image Data

Somiya Metu and Robert P. Winkler
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **REPORT DATE** *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED (From - To)**
September 2013	Final	January to July 2013

4. TITLE AND SUBTITLE
Android REST Client Application to View, Collect, and Exploit Video and Image Data

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Somiya Metu and Robert P. Winkler

5d. PROJECT NUMBER
R0006163.9

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: ARL-CII-B
2800 Powder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-6639

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
As part of the U.S. Army Research Laboratory's image processing testbed, a Representational State Transfer (REST) Web service framework for recording, viewing, and processing video and images has been developed. Client applications are needed to collect data and invoke the Web services. This report describes an Android based client application for collecting, viewing, and exploiting both video and imagery data and initiating image processing algorithms for contrast enhancement, deblurring, and super resolution.

15. SUBJECT TERMS
Android M-JPEG video stream, Android screen annotation, Android REST client

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Somiya Metu
Unclassified	Unclassified	Unclassified	UU	30	**19b. TELEPHONE NUMBER** (*Include area code*) (301) 394-1398

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Listings

1. Introduction

As part of the U.S. Army Research Laboratory's image processing testbed, we developed a Representational State Transfer (REST) Web service framework for recording, viewing, and processing video and images from cameras situated throughout the Adelphi Laboratory Campus (Winkler and Schlesiger, 2013). While the Web services provide the processing and recording capabilities, client applications are required to collect the data and initiate the processing stages. In this report, we describe one such Android application for collecting, viewing, and exploiting both video and imagery data and initiating image processing algorithms for contrast enhancement, deblurring, and super resolution.

2. Development Environment

Google's Android is an open-source software stack intended for mobile devices such as cell phones and tablets. The development environment consists of a software development kit (SDK) written for the Java programming language, and a series of tools integrated with the Eclipse integrated development environment (IDE) for development and debugging. The Android SDK and the associated tools are freely available from Google at http://developer.Android.com. The version used for this development was SDK Tools Revision 10 with the Android Development Tools (ADT) Eclipse plug-in for Eclipse Indigo. The target platform was Android 4.0.3. The Galaxy Nexus phone was used for testing.

Additionally, the Android's Camera application programming interface (API) was used, which provides connectivity to the built-in camera.

3. Application Design

3.1 The Look and Feel

Video Viewer is the Android activity that presents the user interface (UI) for the Android client that communicates with two Web services, namely, the Image Processing Web Service and the Digital Video Recorder (DVR) Web Service. The activity presents the UI through a number of widgets organized in an Android layout. The widgets and layouts are specified in an extensible markup language (XML) format and this information is stored in the application's resource folder.

The main screen provides configuration information for the Web services and a Start button to launch the application with default configurations for the Web services, as shown in figure 1. It also provides an option to change the current configuration by specifying the URLs of the Web services.

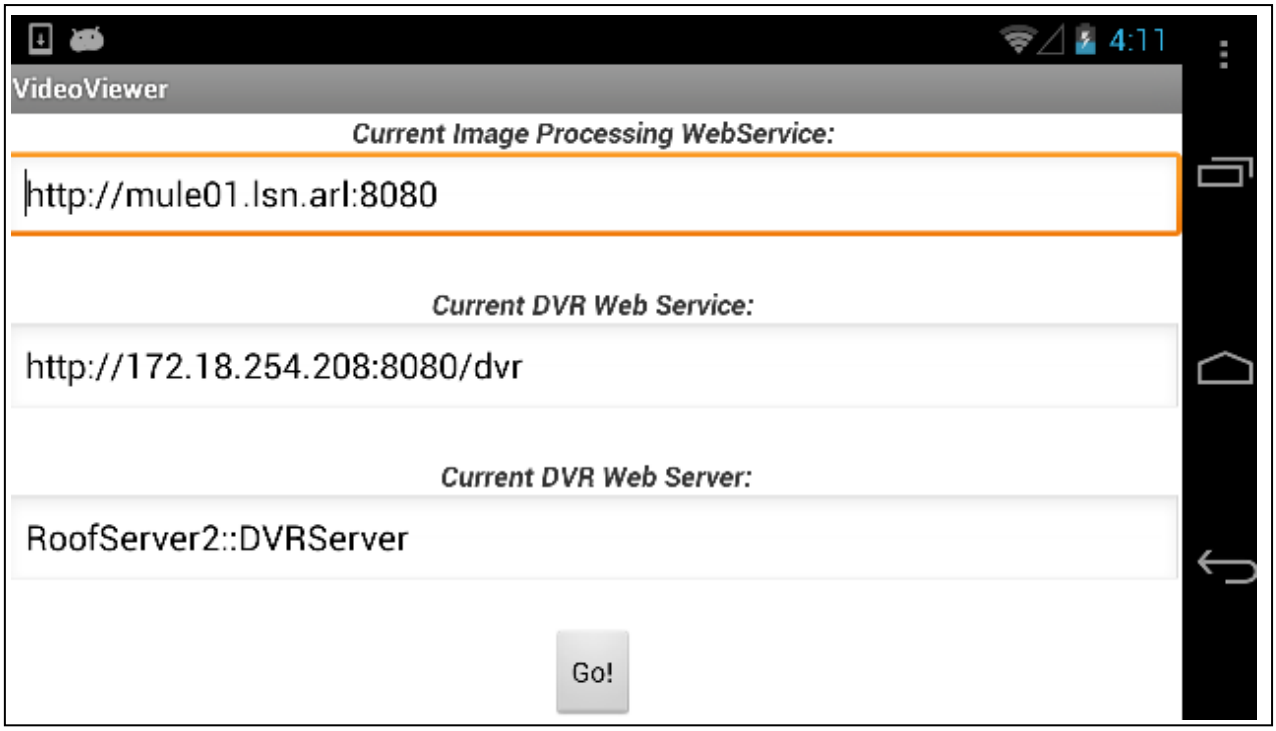

Figure 1. Main screen.

Once the application is launched, a list of cameras available via the DVR Web Service is displayed, as shown in figure 2

Figure 2. Camera list.

Once a camera selection is made, a live-motion JPEG video from the selected camera is displayed, as shown in figure 3. This video has fast-forward, rewind, and pause capabilities. The scrollbar also enables the user to go back to a specific frame/image. The camera list also contains a “Local Camera”. This represents the phone’s internal camera. By choosing this, the phone’s internal built-in camera and its internal image gallery are made available.

Figure 3. Live video display.

3.2 Video Manipulation

3.2.1 Displaying Video

In order to stream a motion JPEG video to an Android device, we have used an open-source Java class from Google code. To start the video display, *MainActivity*, which is a window for placing UI elements, requests a camera list from the DVR Web Service. Once the DVR Web Service responds back with the camera list, a camera can be selected by the user from the list. The *Main Activity* then initializes a *VideoViewer* object by invoking the *start* method and providing the URL of the DVR Web Service as one of the parameters. The *VideoViewer* object then returns the current frame back to the *Main Activity* where it is converted back to a Bitmap object and displayed on the canvas. The *Main Activity* provides the current frame number to the Scrollbar so that it is positioned appropriately to reflect the progress of the video display. This process continues repeatedly, as depicted in the unified modeling language (UML) sequence diagram in figure 4.

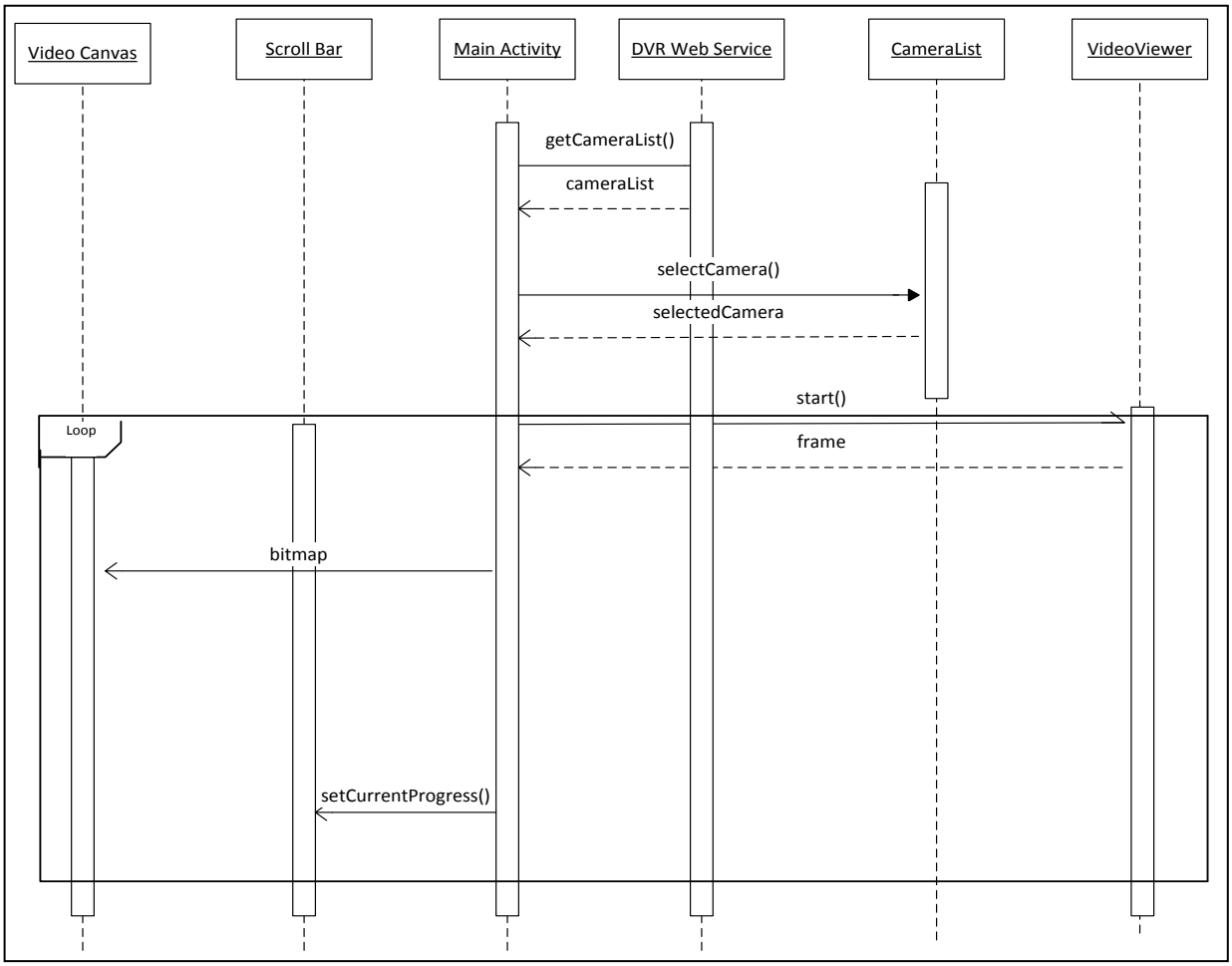

Figure 4. Display video sequence diagram.

3.2.2 Pausing Video

To pause the video, the visibility of the *VideoViewer* object is set to "GONE". This means that the object becomes invisible and does not take up any space for layout. The frame that was paused is converted to a bitmap object and supplied to a newly instantiated *ImageView*. The bitmap is then used to set the content of the *ImageView*. The *ImageView* then renders its content by invoking the draw method. This process continues repeatedly, as depicted in the UML sequence diagram in figure 5.

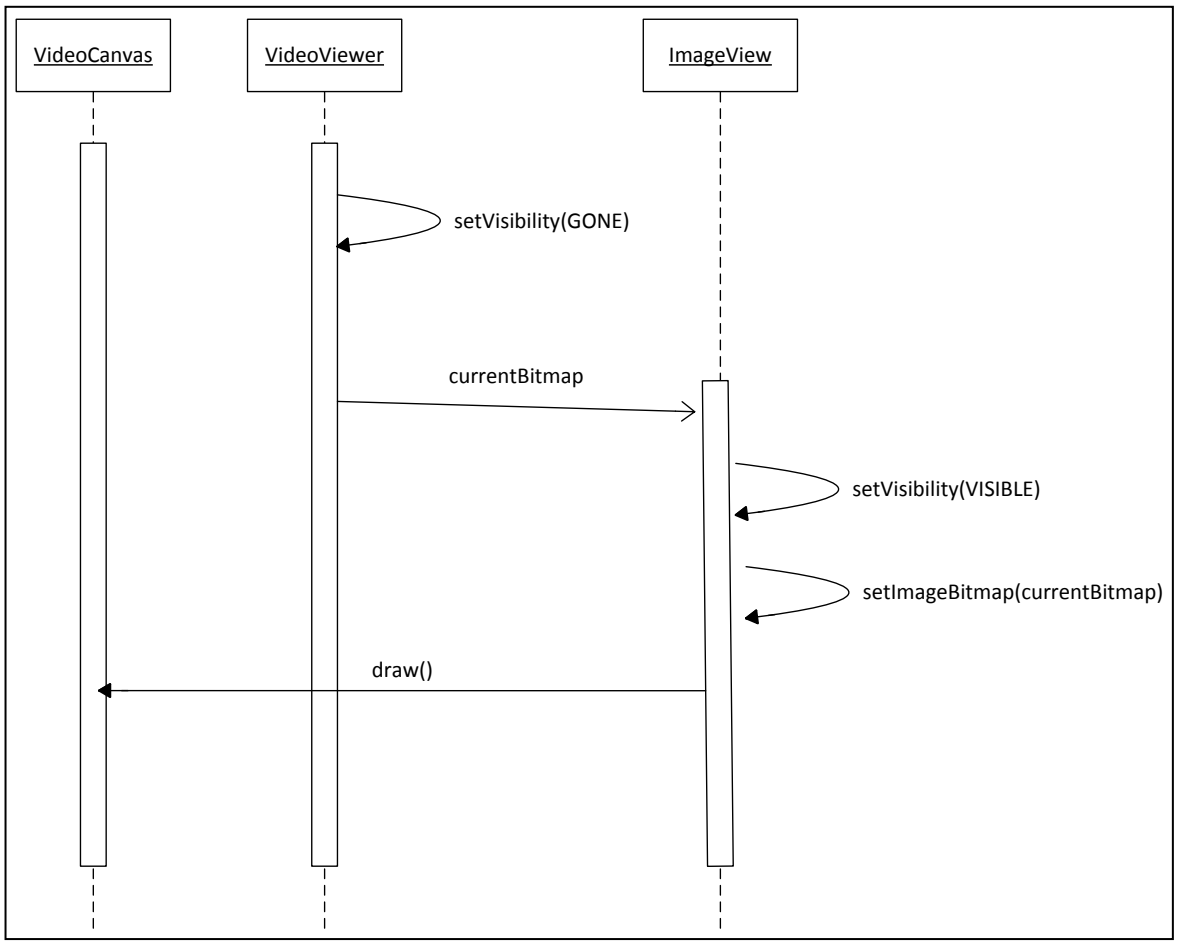

Figure 5. Pause video sequence diagram.

3.2.3 Fast Forward and Rewind

The *Main Activity* destroys the current *VideoViewer* object by invoking the stop feed method. Following this, it initializes a new *VideoViewer* object by invoking the start method. It provides the frame number, fast-forward, or rewind speed along with the URL of the DVR Web Service as its parameters. The frame numbers indicates the frame at which the video has to be fast-forwarded or rewound. The *VideoViewer* object then returns the current frame back to the *Main*

Activity, where it is converted back to a bitmap object and eventually displayed on the canvas. The *Main Activity* provides the current frame number to the scrollbar so that it is positioned appropriately to reflect the progress of the video display. This process continues repeatedly, as depicted in the UML sequence diagram in figure 6.

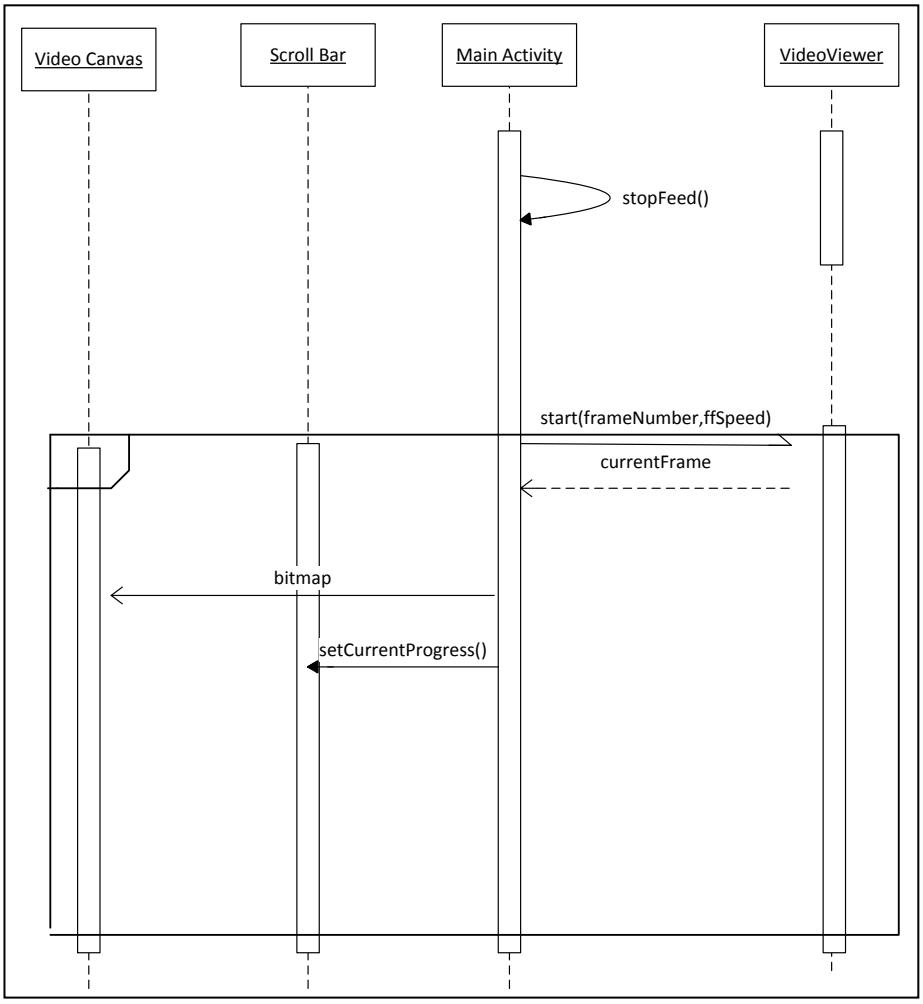

Figure 6. Fast-forward/rewind sequence diagram.

3.3 Image Processing

The live streaming video that is displayed on the device's screen can be paused to initiate image processing tasks on the paused image/frame. The Image Processing Web Service is invoked to perform the image processing tasks on the paused image/frame. The image processing tasks that are currently available are text and graphical image annotation, contrast enhancement, image deblurring, and super resolution. The UI menu for the image processing tasks is shown in figure 7. These image processing tasks are described below.

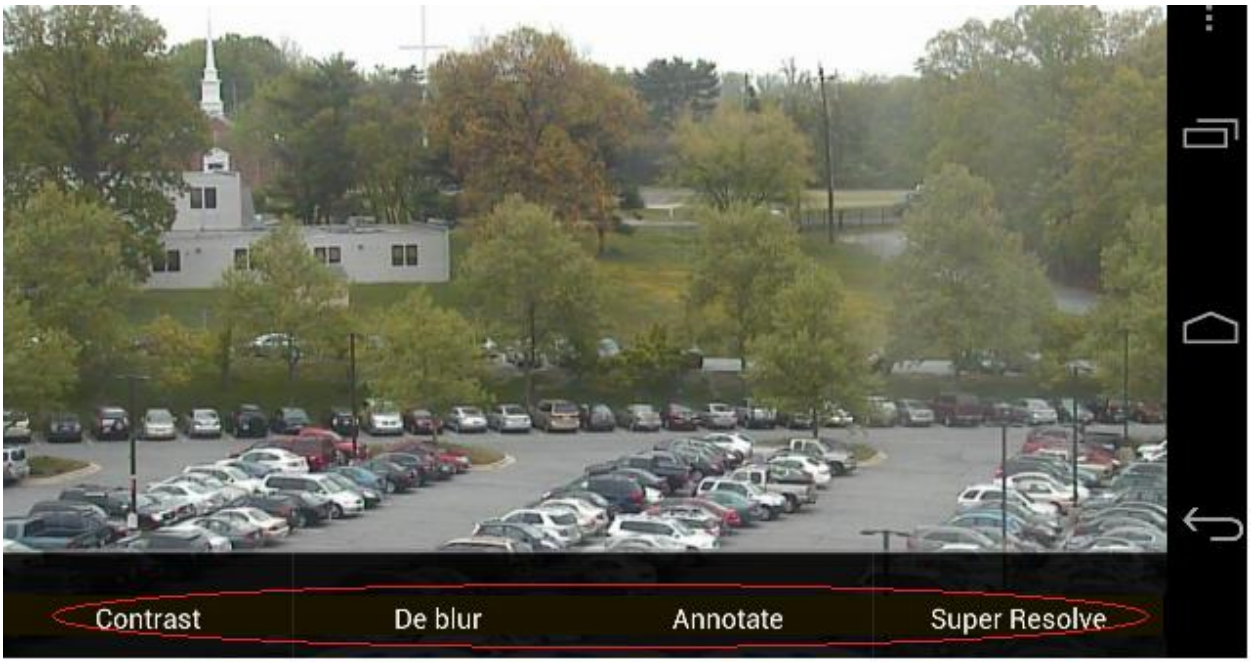

Figure 7. Image processing options.

3.3.1 Annotation

The image can be annotated by drawing on the image or writing text on the image. The annotation feature can be invoked by clicking on the Annotate menu. This opens a list of sub-menu items related to annotation, as shown in figure 8.

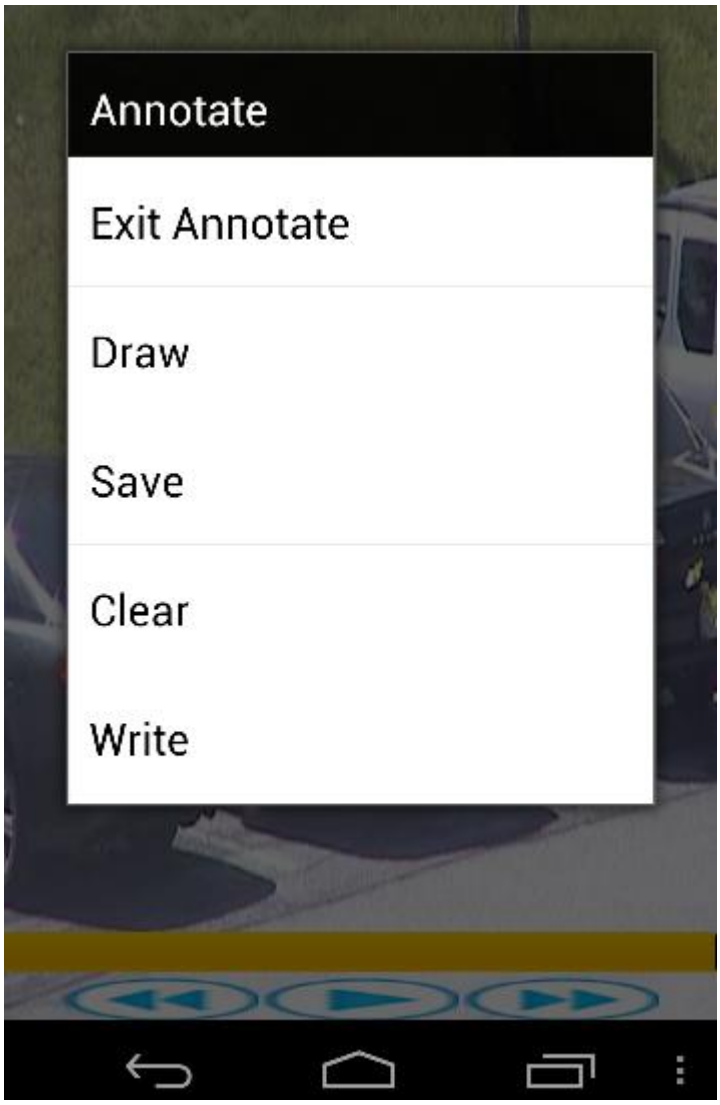

Figure 8. Annotate options.

The annotation feature makes use of an *AnnotateImageView* class. This class extends the *ImageView* class and implements the "OnTouch" event and facilitates drawing based on touch. It contains a private class namely *Point*, which stores the coordinate information that is generated by a "Touch" event. *Point* also stores the text information that the user writes on the image. The *AnnotateImageView* gets instantiated when the Annotation option is selected from the menu. The visibility of the *VideoViewer* object is set to "GONE". This implies that the object becomes invisible and does not take up any space for layout purposes. The background for *AnnotateImageView* is set to the current bitmap object. This view keeps track of all the coordinates on the screen that were touched by the user in a list containing Point objects. Based on the stored coordinates in the Point objects, the drawing is rendered. If the user chooses to save the annotation, the user can save it by choosing Save Annotation from menu. This will save the

image and the annotations in InkML format. This process is depicted in the UML sequence diagram shown in figure 9.

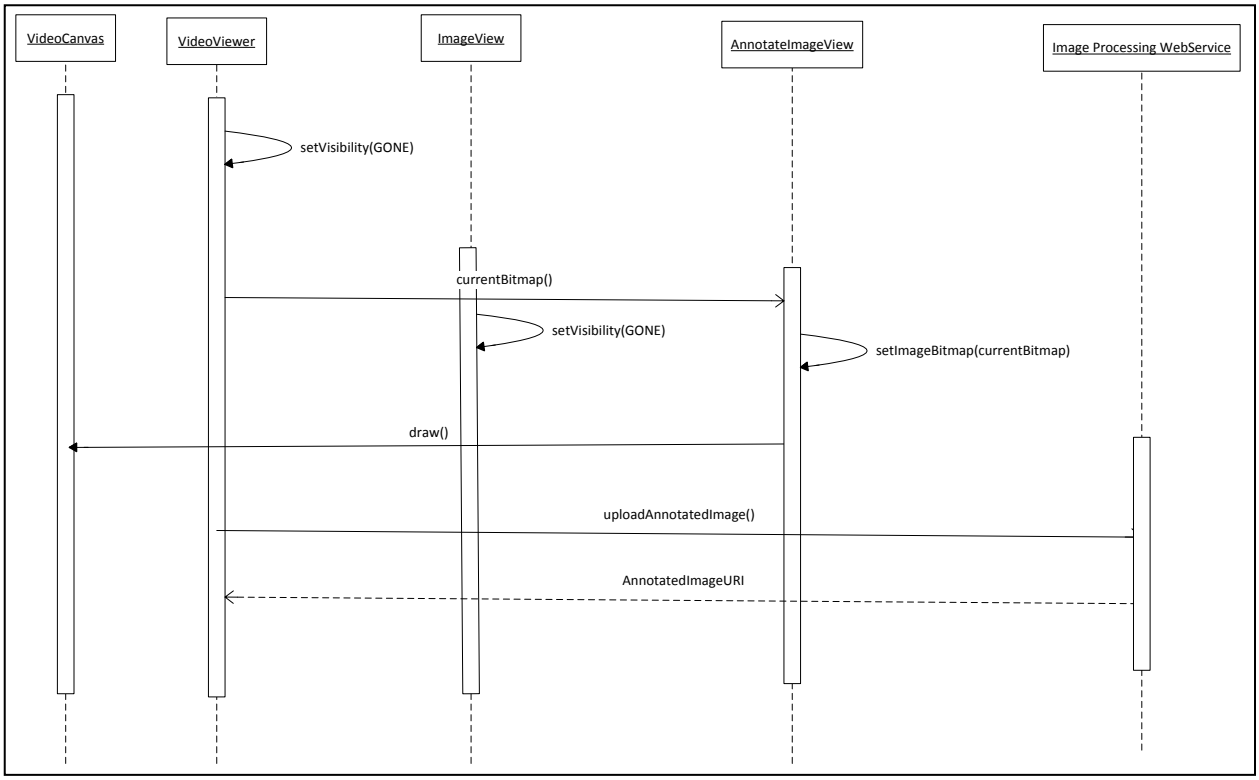

Figure 9. Annotate sequence diagram.

3.3.1.1 Graphical Annotation

The Draw option facilitates free-flow drawing on the image as shown in figure 10. The list of *Point* objects is used to render the drawing.

3.3.1.2 Textual Annotation

The Write option allows users to write text on the image. The user can write the text using the phone soft-keyboard, which appears on the screen when “write” is pressed. Once the user writes the desired text using the keyboard, the “Done” button is pressed. This hides the soft keyboard. Now the text can be dropped on any part of the image just by touching the screen on the area where the user would like the text to appear.

Figure 10. Annotated image.

3.3.1.3 Erasing

The Clear option clears any type of annotation/markings present on the image. The list of Point objects containing tracked coordinate values is cleared and the list becomes empty.

3.3.1.4 Storing

The Save option saves the annotated image and then publishes the annotated image to the Image Processing Web Service. It calls the *saveView* method to convert the bitmap and annotation associated with it to a JPEG file, which gets stored in the device's internal storage. Following this, the *createannotationElement* method is invoked. This method converts the stored annotation information into InkML format. Once the annotation information is coded in InkML language, the *uploadAnnotatedImage* method is invoked. This method is responsible for uploading the image in base64 notation together with its annotation information coded in InkML format. In this method, an asynchronous task is instantiated namely *postServerTask*. This task takes the uniform resource identifier (URI) of the image to be posted and an InkML string, which is a metadata object including an annotation element. An example metadata is shown in listing 1.

```
<?xml version="1.0" encoding="utf-8"?>

<Metadata URI="http://mule01.lsn.arl:8080/ARLImageProcessing/images/4814cbf6-e6e1-4397-
a592-b87149197d10" xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <MediaType>image/png</MediaType>
  <DateCreated>2013-05-06T15:06:44.9048581-04:00</DateCreated>
  <LastModified>2013-05-06T15:06:44.9048581-04:00</LastModified>
  <Annotations>
    <ink xmlns="http://www.w3.org/2003/InkML">
      <definitions>
        <brush xml:id="B1">
          <brushProperty name="width" value="4.0"/>
          <brushProperty name="height" value="4.0"/>
          <brushProperty name="color" value="#FF000000"/>
          <brushProperty name="tip" value="ellipse"/>
        </brush>
      </definitions>
      <trace brushRef="B1">
        485.23962 307.07352, 483.91577 309.55798, 483.23453   318.43945,
        484.54724 331.3474, 492.75012 345.29355, 503.93808 351.06744,
        514.09503 352.63522, 534.7229 350.1277, 545.30304 345.49985,
        549.3976 340.78232, 554.35284 331.53326, 552.2034 324.1612,
        543.755 312.2683, 531.7832 303.72626, 508.53308 296.96783,
        491.88242 298.66687, 487.24475 303.32874
      </trace>
    </ink>
  </Annotations>
  <Media>...image data omitted for brevity...</Media>
</Metadata>
```

Listing 1. XML POST for saving annotated image.

3.3.1.5 Leaving Annotation Mode

The Exit option exits from the Annotation mode.

3.3.2 Super Resolution

Super resolution (*2, 3*) of the paused image/frame is initiated by selecting the Super Resolution option from the menu. In order to super resolve a portion of the image, the user can draw over the image to mark an area of interest that needs to be super resolved. The layout for super resolution is specified in the layout folder under the application's resource folder (called res). It uses a frame layout nested inside a relative layout. Inside the frame layout is one single custom *ImageView*, namely, *SuperResolverImageView*, which has been defined by extending the *ImageView* class from the Android API. We implement a touch event for the *SuperResolverImageView*, which facilitates the user to draw a rectangle in order to select a part of the image for super resolution. Figure 11 depicts this selection.

Figure 11. Selected area for super resolution.

To super resolve the image, the current frame, as well as several frames following the current frame, are collected as required by the Image Processing Web Service. The preferred number of frames collected to pass on to the Web Service is five. To achieve this, an asynchronous task *BitmapCollection* is invoked. Once the task is complete, it returns an array list containing bitmap objects. After the list is created and the area of the image to be super resolved is determined by the user, an asynchronous task *SuperResolveAsyncTask* is initiated.

The execute method is called from the asynchronous task instance. This method runs on the main UI thread, which triggers a worker thread. Before handing the control to the worker thread, *onPreExecute* method is called. In this method, we show the dialog indicating that the super resolution has been initiated. Following the *onPreExecute* method, the *doInBackground* method is called where the bulk of the work is performed. The array list of bitmaps that was collected earlier is traversed one by one to be uploaded to the Image Processing Web Service. To achieve this, the *uploadImage* method is invoked to upload the image data one by one. In this method, the bitmap data are converted into a base64 string. An XML string encapsulating the media and header information is created. The XML string conforms to the metadata.xsd schema specified by the REST Web Service. This schema represents the metadata associated with an image file. The elements in the XML string that get set are “MediaType” and “Media”.

An example of the Metadata is shown in listing 2.

```
<?xml version="1.0" encoding="utf-8"?>
<Metadata URI="http://mule01.lsn.arl:8080/ARLImageProcessing/images/21446696-ad39-
4a7d-abbc-042f2381c917" xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <MediaType>image/jpeg</MediaType>
      <DateCreated>2013-05-02T15:19:18.310355804:00</DateCreated>
      <LastModified>2013-05-02T15:19:18.3103558-04:00</LastModified>
      <Media>…image data omitted for brevity… </Media>
</Metadata>
```

Listing 2. XML image POST.

Following the creation of the metadata object listed above, the *postToServer* method is invoked. This method takes two parameters. The first one is the URI of the image to be posted and the second one is the metadata object. In this method, an HTTP client and HTTP post instance is created. The HTTP post object is encoded with header information and metadata. The HTTP post then gets executed using the HTTP client created before. In response to the HTTP request, the Image Processing Web Service sends back the URI of the uploaded image. This URI is then saved to an array list, which eventually contains the URIs of all the frames that have been successfully uploaded. The *uploadCollection* method is then invoked. This method is responsible for creating an XML string that conforms to a schema named collection.xsd. This schema is for a collection of images/frames as specified by the REST Web Service. The elements in the XML string that get set are "Title" and "ImageURI". The XML string contains all the header information and the URI of all the uploaded frames/images on the Web Service.

An example of collection metadata is shown in listing 3.

```
<?xml version="1.0" encoding="utf-8"?>
<Collection URI="http://mule01.lsn.arl:8080/ARLImageProcessing/collections/f51413f5-badc-
47bb-a3f6-89076185964f" xmlns="http://arl.army.mil/ARLImageProcessing/Collection.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <Title>Camera frames :  1367522366504 </Title>
  <DateCreated>2013-05-02T15:19:26.8322927-04:00</DateCreated>
  <ImageURI>http://mule01.lsn.arl:8080/ARLImageProcessing/images/21446696-ad39-4a7d-
abbc042f2381c917</ImageURI>
  <ImageURI>http://mule01.lsn.arl:8080/ARLImageProcessing/images/aaaf6336-1144-4809-b23c-
242a3018ccf0</ImageURI>
  <ImageURI>http://mule01.lsn.arl:8080/ARLImageProcessing/images/3c615f22-b922-437b-ae31-
4e4b15c39fd6</ImageURI>
  <ImageURI>http://mule01.lsn.arl:8080/ARLImageProcessing/images/3f5557e3-7c8f-4e81-98c5-
27cd415229c4</ImageURI>
  <ImageURI>http://mule01.lsn.arl:8080/ARLImageProcessing/images/4815b519-3efe-497d-aead-
3ef4902cf7d8</ImageURI>
</Collection>
```

Listing 3. XML collection POST.

Once the Collection object is formulated, then the *postToServer* method is invoked. This method takes the URI of the collection to be posted and the XML string representing the collection. This method returns the URI of the uploaded Collection object. Following this, the *superResolve* method is invoked by passing the URI of the uploaded Collection object. In this method, an XML string conforming to the metadata.xsd schema is created. The elements of the schema that get set are "ParentURI", "Chip", and "SuperResolutionParameters". The value of the ParentURI element points to URI of the uploaded collection. The value of the chip represents the bounding box of the selected area. The SuperResolutionParameters, as the name suggests, holds the values for the parameters for super resolution. An example of the metadata is shown in listing 4.

```
<Metadata
URI="http://mule01.lsn.arl:8080/ARLImageProcessing/collections/superresolved/fcf0dfc8-
8137-458d-956a-eb539080de63">
  <MediaType xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd">
    image/png
  </MediaType>
  <DateCreated xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd">2013-05-
02T08:34:21.7512257-04:00</DateCreated>
  <LastModified xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd">2013-05-
02T08:34:21.7512257-04:00</LastModified>
  <ParentURIxmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd">
    http://mule01.lsn.arl:8080/ARLImageProcessing/collections/966085ab-4c70-4bec-b8cf-
06f576c47230</ParentURI>
    <Chip LowerRightX="346" LowerRightY="255" UpperLeftX="278" UpperLeftY="208"
xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd"/>
    <SuperResolutionParameters Align="true" Inset="5" Quality="10" Speed="0.55"
xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd"/></Metadata>
```

Listing 4. XML super resolution POST.

Once the XML string is formulated, the *postToServer* method is invoked by passing the URI for the Super Resolution service and the XML string. This method returns the URI of the super-resolved image.

The *getResult* method is then invoked by passing the URI of the super-resolved image.

In this method, the *HTTPGet* instance is created using the URI of the super-resolved image residing on the Image Processing Web Service. A *DefaultHTTPClient* instance is created and then *HTTPGet* is executed using the HTTP client. An HTTP response is then generated by the client, which is processed to obtain the super-resolved image in the form of a base64 string. The *getResult* method then returns the super-resolved image as a base64 string. This string is then converted to bitmap format. Following this, the *runOnUiThread* method is invoked by passing a runnable action as a parameter. The runnable action in this case is *UpdateImage*. The *UpdateImage* action runs on the main UI thread. It sets the *SuperResolverImageView* to the

super-resolved image in bitmap format. The super-resolved image is then visible on the screen as shown in figure 12.

Figure 12. Super-resolved result.

The UML sequence diagram in figure 13 depicts the super-resolution process.

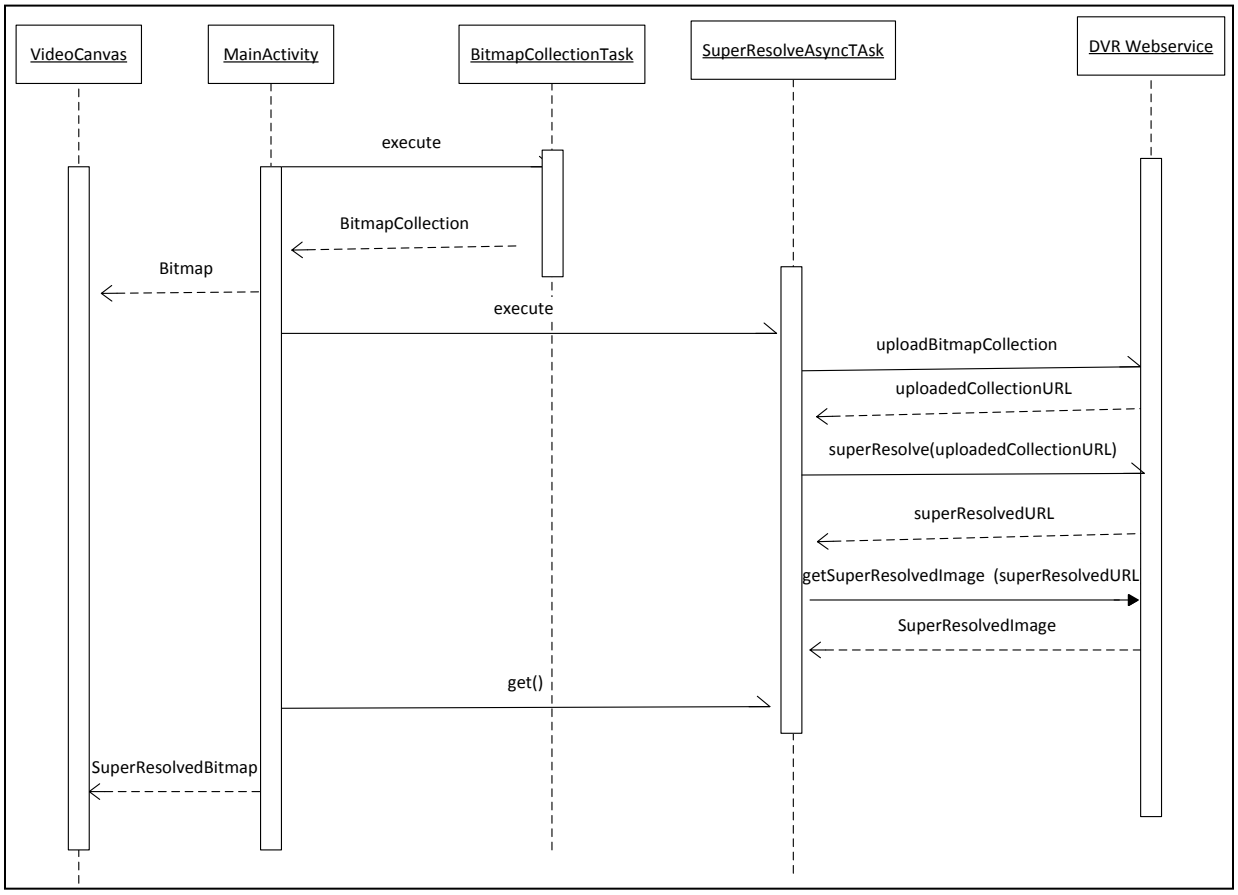

Figure 13. Super resolution sequence diagram.

3.3.3 Contrast and Deblur

The Contrast (*4*) or Deblur (*5*) image task is applied to the paused image/frame by clicking on the Contrast or Deblur option from the menu, respectively. This invokes the *saveFrameForProcessing* method. This method saves the current frame to the device's internal storage in PNG format. The saved file is then decoded to a bitmap object. Following this, an asynchronous task, *ProcessImageAsyncTask*, is initiated. The execute method is called from the asynchronous task instance. This method runs on the main UI thread, which triggers a worker thread. Before handing the control to the worker thread, the *onPreExecute* method is called. This method passes setup values from the main UI thread to the worker thread. In this case, the dialog indicating that the image processing has commenced is displayed. Following this, the *doInBackground* method is called on the worker thread. This method is where the bulk of the work is performed. The *UploadImage* method is called to load the selected image on the server. The bitmap data are converted into a base64 string. An XML string encapsulating the media and header information is created. The XML string conforms to the metadata.xsd schema specified by the REST Web Service. This schema represents the metadata associated with an image file.

The elements in the XML string that get set are “MediaType” and “Media”. In this method, the image, which is in bitmap format, is encoded in base64 notation to be sent to the server.

Once the image is uploaded to the server, the server sends back a URI indicating the location of the uploaded image on the server. Based on the user selection of Contrast or Deblur the corresponding method, *requestContrast* or *requestDeblur* method, is invoked. In these methods, an XML string conforming to the metadata.xsd schema is generated. The XML string contains parameter values for the Contrast/Deblur Service. An example of the metadata with contrast parameters is shown in listing 5.

```
<?xml version="1.0" encoding="utf-8"?>
<Metadata
URI="http://mule01.lsn.arl:8080/ARLImageProcessing/images/contrastenhanced/6b6217bb-
5832-4b29-9f56-60ac5db464ae"
xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<MediaType>image/png</MediaType><DateCreated>2013-05-02T15:26:06.6286398-
04:00</DateCreated><LastModified>2013-05-02T15:26:06.6286398-
04:00</LastModified><ParentURI>http://mule01.lsn.arl:8080/ARLImageProcessing/images/0
f6793a0-6a5e-4402-9e17-9a8039e12cc1</ParentURI><ContrastEnhancementParameters
PercentageLow="0.5" PercentageHigh="0.002"/><Media>image data omitted for brevity
</Media>
</Metadata>
```

Listing 5. XML contrast POST.

An example of the metadata with deblur parameters is shown in listing 6.

```
<?xml version="1.0" encoding="utf-8"?><Metadata
URI="http://mule01.lsn.arl:8080/ARLImageProcessing/images/deblurred/2f9db5d0-e41c-
4341-a941-e1aa563ba495" xmlns="http://arl.army.mil/ARLImageProcessing/Metadata.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<MediaType>image/png</MediaType>
<DateCreated>2013-05-02T15:26:11.1294958-04:00</DateCreated>
<LastModified>2013-05-02T15:26:24.466145-04:00</LastModified>
<ParentURI>http://mule01.lsn.arl:8080/ARLImageProcessing/images/contrastenhanced/6b62
17bb-5832-4b29-9f56-60ac5db464ae</ParentURI><DeblurParameters Alpha="0.3" MG="20"/>
<Media>… image data omitted for brevity…</Media>
</Metadata>
```

Listing 6. XML Deblur POST.

Once the XML string is formulated, the *postToServer* method is invoked, which takes the URI of the Contrast or Deblur service and the XML string representing the metadata.

An HTTP request is made to the REST Web Service for contrast/deblur. An HTTP client and HTTP POST instance is instantiated. The HTTP POST data are encoded with the XML string and is executed using the HTTP client. In response to the HTTP request, the Image Processing Web Service sends back the processed image in base64 notation, which is then converted back in bitmap format. The *SuperResolverImageView* is then made visible on the screen and its content is set to the resulting bitmap displaying the contrasted/deblurred image. Figures 14 and 15 depict the contrasted and deblurred image, respectively.

Figure 14. Contrasted image.

Figure 15. Deblurred image.

The sequence diagram shown in figure 16 depicts the contrast/deblur process.

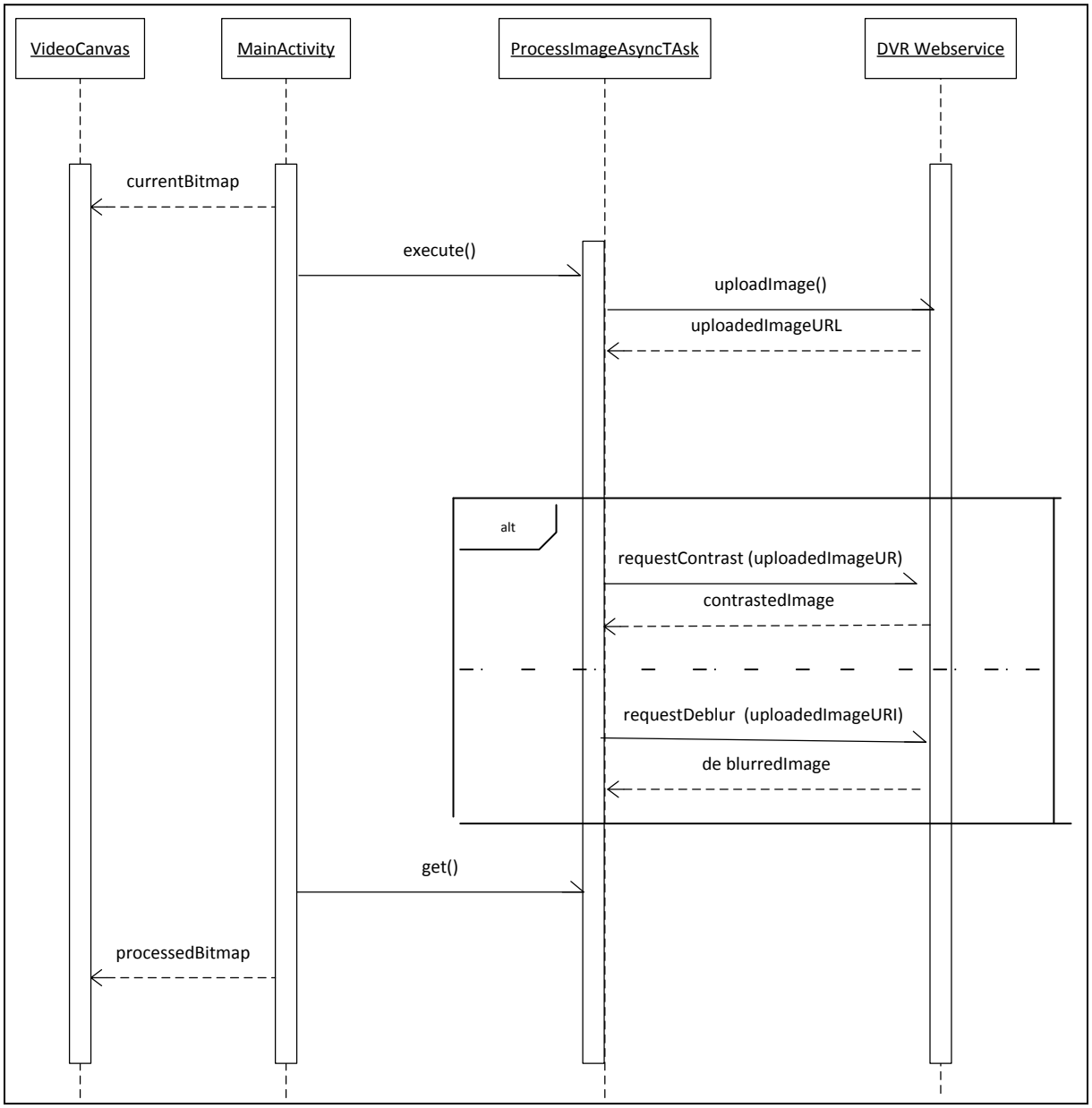

Figure 16. Contrast/deblur sequence diagram.

4. Conclusion and Future Work

In this report, we have described an Android application for collecting, viewing, manipulating, and exploiting video and image data. The application initiates image processing algorithms for contrast enhancement, deblurring, and super resolving these data by communicating with the REST Web Services that form the foundation of our image processing testbed. As more image processing algorithms are added to the Android testbed, this application will be extended to invoke those algorithms.

5. References

1. Winkler, R. P.; Schlesiger, C. *Image Processing REST Web Services*; ARL-TR-6393; U.S. Army Research Laboratory: Adelphi, MD March 2013.

2. Young, S. S.; Driggers, R. G. Superresolution Image Reconstruction From a Sequence of Aliased Imagery. *Applied Optics* **2006**, *45* (21), 5073–5085.

3, Driggers, R. G.; Krapels, K. A.; Murrill, S.; Young, S. S.; Theilke, M.; Schuler, J. M. Superresolution Performance for Undersampled Imagers. *Optical Engineering* **2005**, *44* (01).

4. Young, S. S.; Driggers, R. G. *Signal Processing and Performance Analysis for Imaging Systems*. Artech House, 2008.

5. Young, S. S.; Driggers, R. G.; Teaney, Brian P.; Jacobs, Eddie L. Adaptive Deblurring of Noisy Images. *Applied Optics* **2007**, *46* (5), 744–752.

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	GOVT PRINTG OFC A MALHORTA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB IMAL HRA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL
3 (PDS)	DIRECTOR US ARMY RESEARCH LAB RDRL-CII-B L TOKARCIK R WINKLER S METU
4 (PDFS)	RDRL-CII-T S LAROCCA J MORGAN M HOLLAND M VANNI
3 (PDFS)	RDRL-CII-C M THOMAS M MITTRICK J RICHARDSON
1 (PDF)	DIRECTOR NATIONAL SECURITY AGENCY CAMT M MARTINDALE
1 (PDF)	DIRECTOR ODNI FLPO J KLAVENS
1 (PDF)	DIRECTOR HQDA ODCS, G2 CSM M SANCHEZ
1 (PDF)	DIRECTOR CERDEC RDER-CPM-IM R SCHULZE
1 (PDF)	DIRECTOR INSCOM R RICHARDSON

INTENTIONALLY LEFT BLANK.